FORM PTO-1390 (REV. 12-2001) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371

ATTORNEY 'S DOCKET NUMBER

U.S. APPLICATION NO. (If known, see 37 CFR 1.5)
10/069297

INTERNATIONAL APPLICATION NO.	INTERNATIONAL FILING DATE	PRIORITY DATE CLAIMED
PCT/EP00/07811	21 July 2000	

TITLE OF INVENTION

APPLICANT(S) FOR DO/EO/US

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☒ This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.
2. ☐ This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. ☐ This is an express request to begin national examination procedures (35 U.S.C. 371(f)). The submission must include items (5), (6), (9) and (21) indicated below.
4. ☒ The US has been elected by the expiration of 19 months from the priority date (Article 31).
5. ☒ A copy of the International Application as filed (35 U.S.C. 371(c)(2))
   a. ☒ is attached hereto (required only if not communicated by the International Bureau).
   b. ☐ has been communicated by the International Bureau.
   c. ☐ is not required, as the application was filed in the United States Receiving Office (RO/US).
6. ☐ An English language translation of the International Application as filed (35 U.S.C. 371(c)(2)).
   a. ☐ is attached hereto.
   b. ☐ has been previously submitted under 35 U.S.C. 154(d)(4).
7. ☐ Amendments to the claims of the International Aplication under PCT Article 19 (35 U.S.C. 371(c)(3))
   a. ☐ are attached hereto (required only if not communicated by the International Bureau).
   b. ☐ have been communicated by the International Bureau.
   c. ☐ have not been made; however, the time limit for making such amendments has NOT expired.
   d. ☐ have not been made and will not be made.
8. ☐ An English language translation of the amendments to the claims under PCT Article 19 (35 U.S.C. 371 (c)(3)).
9. ☒ An oath or declaration of the inventor(s) (35 U.S.C. 371(c)(4)).
10. ☐ An English lanugage translation of the annexes of the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

Items 11 to 20 below concern document(s) or information included:

11. ☐ An Information Disclosure Statement under 37 CFR 1.97 and 1.98.
12. ☐ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.
13. ☐ A FIRST preliminary amendment.
14. ☐ A SECOND or SUBSEQUENT preliminary amendment.
15. ☐ A substitute specification.
16. ☐ A change of power of attorney and/or address letter.
17. ☐ A computer-readable form of the sequence listing in accordance with PCT Rule 13ter.2 and 35 U.S.C. 1.821 - 1.825.
18. ☒ A second copy of the published international application under 35 U.S.C. 154(d)(4).
19. ☐ A second copy of the English language translation of the international application under 35 U.S.C. 154(d)(4).
20. ☐ Other items or information:

U.S. APPLICATION NO. (if known, see 37 CFR 1.5)	INTERNATIONAL APPLICATION NO.	ATTORNEY'S DOCKET NUMBER
10/069297		

21. ☒ The following fees are submitted:	CALCULATIONS PTO USE ONLY
BASIC NATIONAL FEE (37 CFR 1.492 (a) (1) - (5)):	
Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO **$1040.00**	
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO **$890.00**	
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO **$740.00**	
International preliminary examination fee (37 CFR 1.482) paid to USPTO but all claims did not satisfy provisions of PCT Article 33(1)-(4) **$710.00**	
International preliminary examination fee (37 CFR 1.482) paid to USPTO and all claims satisfied provisions of PCT Article 33(1)-(4) **$100.00**	
ENTER APPROPRIATE BASIC FEE AMOUNT =	$ 890.00
Surcharge of **$130.00** for furnishing the oath or declaration later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492(e)).	$

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE	
Total claims	- 20 =		x **$18.00**	$
Independent claims	- 3 =		x **$84.00**	$
MULTIPLE DEPENDENT CLAIM(S) (if applicable)			+ **$280.00**	$

TOTAL OF ABOVE CALCULATIONS =	$
☒ Applicant claims small entity status. See 37 CFR 1.27. The fees indicated above are reduced by 1/2. +	$
SUBTOTAL =	$ 445.00
Processing fee of **$130.00** for furnishing the English translation later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492(f)).	$
TOTAL NATIONAL FEE =	$
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). **$40.00** per property +	$ 40.00
TOTAL FEES ENCLOSED =	$ 485.00
Amount to be refunded:	$
charged:	$ 485.00

The PTO did not receive the following listed item(s)
Assignment form

a. ☐ A check in the amount of $ _____ to cover the above fees is enclosed.

b. ☐ Please charge my Deposit Account No. _____ in the amount of $ _____ to cover the above fees. A duplicate copy of this sheet is enclosed.

c. ☐ The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. _____. A duplicate copy of this sheet is enclosed.

d. ☒ Fees are to be charged to a credit card. **WARNING:** Information on this form may become public. **Credit card information should not be included on this form.** Provide credit card information and authorization on PTO-2038.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137 (a) or (b)) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:

AnteVista GmbH
Harburger Schlossstrasse 6-12
D-21079 Hamburg
Germany

SIGNATURE

Dr. Jean-Paul Theis
NAME

REGISTRATION NUMBER

A microprocessor having an instruction format containing explicit timing information.

1. Field of the invention

The invention is dealing with instruction formats of microprocessors.

2. Conventions, definition of terms, terminology

In the context of the present invention, the term 'microprocessor' means also a central processing unit (CPU) or a digital signal processor (DSP), the meaning of these terms being the one commonly described in the literature. As usual, a microprocessor has an instruction set. In other words, the machine code of a program which is running or executed on said microprocessor, contains exclusively instructions belonging to said instruction set. Said machine code is obtained either by compiling the source code of said program or by manual writing. Each instruction of a said instruction set has an instruction format. As usual, the term 'instruction format' refers to a sequence of bit fields of a certain length. Said bit fields may be of different length. A minimum set of bit fields making up an instruction format normally contains a so called 'opcode' bit field and one or more 'operand' bit fields. Figure 1 illustrates the discussed concepts. The 'opcode' bit field encodes (allows to uniquely identify) a specific instruction, e.g. the addition of two numbers, among all the instructions of said instruction set. The 'operand' bit fields uniquely determine the operands of the instruction encoded in the 'opcode' bit field. In other words, an instruction is a data operation, where the operation is given by (encoded in) the 'opcode' bit field and where the data are given by (encoded in) the 'operand' bit fields. Usually, the operands are either given by memory references, e.g. data stored at some memory addresses, or by contents of registers in which case the registers are uniquely identified by (encoded in) said 'operand' bit fields. E.g. in case of a microprocessor with a register file containing 128 registers, an 'operand' bit field of at least 7 bits is required to uniquely identify (encode) a specific register inside the register file. Furthermore, one distinguishes normally between source operands and destination operands. Usually, source operands represent either memory references or registers containing the data required by an instruction, whereas destination operands represent either memory references or registers to which the result of an instruction, e.g. the addition of two numbers, has to be stored.

In the context of the present invention, the length and the order of the bit fields making up the format of an instruction is not relevant. In other words, it doesn't matter whether the 'opcode' bit field is preceding the 'operand' bit fields or vice versa nor does the order of the 'operand' bit fields among each other

indirect addressing, offset addressing etc..., whereas the 'address' bit field determines the address of the considered operand within a memory space.

However, none of today's instruction formats contains a bit field encoding explicit timing information, where said timing information explicitly determines instruction scheduling and execution as defined before. This lack of information is due to the fact that the architecture concepts of today's microprocessors (CPUs and DSPs as well) doesn't require this type of information because instruction scheduling is done either (1) in case of super-scalar and multi-issue microprocessors (CPUs and DSPs as well), by dynamic scheduling mechanisms based on data dependence analysis of instructions contained in a more or less large instruction window of the compiled or hand written machine code of a given program or (2) in case of VLIW processors by static scheduling techniques, in particular by software pipelining and trace scheduling, such that instructions are scheduled and executed in the same order in which they are arranged in the machine code, where said machine code is generated by applying said static scheduling techniques or (3) in case of EPIC processors, e.g. the IA-64 from Intel Corporation, by a mixture of the approaches (1) and (2). In this sense, timing information contained (encoded) in the instruction format appears to be just redundant information and only likely to increase the machine code size. However this does not hold in conjunction with a static instruction scheduling technique called software pipelining, as will be shown in section 5.

4. Brief description of the drawings

Figure 1 shows an example of a 'conventional' instruction format containing bit fields for 'opcode' and 'operands'.
Figure 2 shows an example of an instruction format containing a bit field containing explicit timing information.
Figure 3 shows a 'for'-loop and the directed acyclic graph ('dag') which equivalently represents the loop body of said 'for'-loop. Nodes of said 'dag' represent instructions of an instruction set of a microprocessor and where said 'dag' is 'software pipelined' with an initiation interval of 1 clock cycle.

explicit timing information for a given instruction contains the integers 2, 3 and 5. This would mean that said instruction would

(a) enter the 'fetch' stage with a delay of 2 clock cycle units with respect to 'time zero', where 'time zero' is the point in time or the clock cycle when the instruction would enter the 'fetch' stage in the absence of any delay information
(b) enter the 'decode' stage 3 clock cycles after having entered the 'fetch' stage
(c) enter the 'execute' stage 5 clock cycles after having entered the 'decode' stage.

As one can see, the timing information, given in form of positive integers, represents delays (in clock cycle units) according to which the entrance points of an instruction into the different pipeline stages have to be delayed with respect to points in time at which said instruction entered the previous pipeline stage. As explained before, the entrance point into the first pipeline stage is thereby delayed with respect to 'time zero', where 'time zero' is the point in time at which said instruction would enter the first pipeline stage in the absence of any timing (delay) information. Using a different terminology, one simply says that the entrance points must be delayed by the delays as given by the integer values contained in the timing information bit field of the instruction format. Therefore, it is assumed that the microprocessor contains some mechanism or hardware circuitry to delay the entrance points of an instruction into each pipeline stage individually. However, it is not relevant for the scope of the present invention how this mechanism is implemented, whether the delays are generated by stalls of the instruction pipeline or by some other method. In the previous example 'incremental timing' encoding was used, in other words the entrance point of an instruction into a certain pipeline stage is determined by adding the delay (as given by the integer value) to the entrance point into the previous pipeline stage. For the scope of the present invention, it must be noted that the method of delaying the entrance point of an instruction into a certain pipeline stage is equivalent to leaving the entrance point unchanged and delaying the point in time at which the instruction 'leaves' said pipeline stage, which is equivalent to increasing the latency of said pipeline stage, where the latency of a pipeline stage can be defined as the number of clock cycles that an instruction takes in order to go through said pipeline stage.

To problem (2) : As mentioned before, the timing information contained in the corresponding bit field of the instruction format may contain timing information for each pipeline stage of a given instruction. Although it is not relevant for the scope of the present invention, two basic encoding schemes are of practical interest and shall be briefly considered. Of course, there exists a myriad of encoding techniques allowing to further compress the timing information by minimizing the redundancy. This however always requires some decoding overhead prior to actual instruction scheduling and execution and usually implies some loss in overall processing speed performance as well as additional power consumption. The two mentioned encoding schemes which shall be considered here are : (a) 'absolute timing' (b) 'incremental timing'. 'Incremental timing' encoding has been used in the previous example. If 'absolute timing' encoding would be used instead, then said bit field would contain the integers 2, 5 (=2+3) and 10 (=2+3+5) respectively and all timing information would be with respect to the time reference (time 'zero') of said instruction, in other words the 'decode' stage would be entered 5 clock

the present discussion to specify the loop body in form of a graphical representation, namely in form of a directed acyclic graph (abbreviated by 'dag' in the following) whose nodes represent instructions of the instruction set of a given microprocessor. A directed edge emanating from a node v_1 and ending at a node v_2 means that node v_2 has to be scheduled and executed **after** node v_1. The presence of 3 nested 'if-then-else' statements in the loop body of the 'for'-loop translates into 3 'compare' instructions in the 'dag' and results in 4 possible branches such that one of the nodes labeled a, b, c or d in figure 3 are executed depending on the outcome of the 'compare' nodes labeled e, f and g. Assuming that there are no data dependencies between iterations of the 'for'-loop, the goal is now to maximize instruction level parallelism and to overlap the scheduling and execution of the different iterations of the 'for'-loop by applying software pipelining and determining the minimum initiation interval. Furthermore, assume that the resource constraints of the microprocessor are such that no more than three instructions can be scheduled and executed at the same time (in the same clock cycle). Neglecting any additional constraints due to operand (register) lifetimes, one can easily verify that the minimum initiation interval is 1 clock cycle long. Furthermore, the 'dag' shown in figure 3 is such that no instruction has to be delayed. However, since the 'dag' is software-pipelined with a period of one clock cycle, 3 independent 'compare' instructions have to be scheduled in one clock cycle leading to 2^3 possible combinations containing each 4 instructions taken from different iterations of the 'for'-loop, namely the combinations : (a,e,g,h), (a,f,g,h), (b,e,g,h), (b,f,g,h), (c,e,g,h), (c,f,g,h), (d,e,g,h), (d,f,g,h). This means that the final machine code corresponding to the 'software-pipelined' 'for'-loop contains at least $2^3 \times 4 = 32$ instructions (it effectively contains even more because additional 'branch' instructions must be inserted in the machine code), which is 4 times more than the number of instructions of a sequential machine code version of the 'for'-loop. Indeed, said sequential machine code version would contain only as many instructions as contained in the 'dag' under the assumption that predicated instructions would be used.

However, by using an instruction format with explicit timing information one can reduce the machine code size obtained by software-pipelining the considered 'dag' or 'for'-loop to the same number of instructions as required by said sequential machine code version. Indeed, it is enough

(1) to indicate the initiation interval of said 'dag' (e.g. of said 'for'-loop body)

(2) to indicate for each instruction (node) in the 'dag', the delay of that instruction such that all resource constraints are satisfied

This is enough information for an appropriately designed microprocessor to schedule and execute all instructions such that said 'for'-loop is effectively software-pipelined with the prescribed initiation interval. Although in the example of figure 3 all the delays are zero, it is easy to figure out this mechanism for the case where the delays are non-zero. As already mentioned previously, delays will usually use 'incremental timing' encoding. In other words, if the entrance point of a node (instruction) with no incoming edges is determined to be at some point on said time axis by taking into account the delay associated to that node, then any node v_2 having a associated delay d_2 and having an incoming edge emanating from some node v_1, where node v_1 entered the first pipeline stage at some clock cycle t (on said time axis) after taking into account the delay associated to that node, is then scheduled to enter the first pipeline stage at clock cycle $t+d_2+1$.

6. Summary of the invention

The present invention concerns a microprocessor having an instruction format containing explicit timing information according to claim 1.

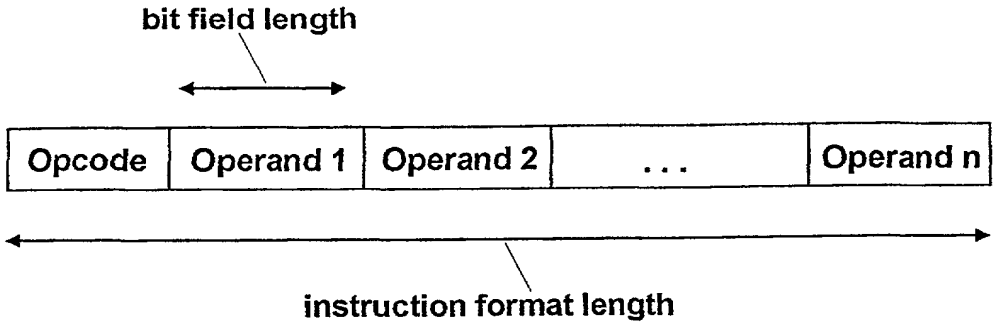


Figure 1

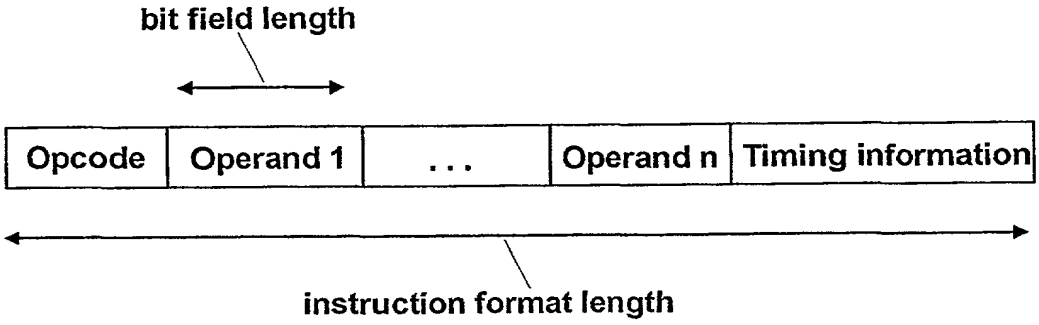
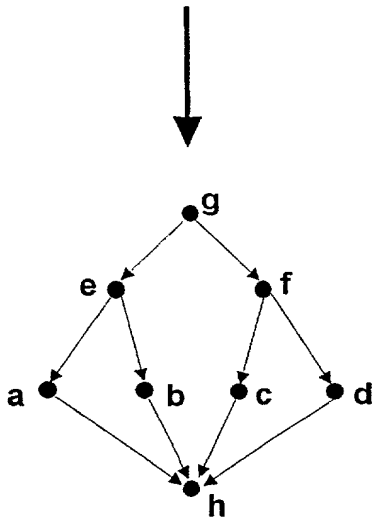


Figure 2

```
for (i==0;i<=N;i++)
{
if ( x[i]<=y[i] ) { if ( x[i+10]>=y[i] ) x[i]=y[i];
                    else x[i]=x[i+3]; }
  else { if ( x[i+3]>=y[i] ) x[i]=y[i+3];
         else x[i]=x[i+5]; }
}
```

loop body of the 'for'-loop



'dag' equivalently representing the loop body of the above 'for'-loop

Figure 3

A microprocessor having an instruction format containing explicit timing information.

1. Field of the invention

The invention is dealing with instruction formats of microprocessors.

2. Conventions, definition of terms, terminology

In the context of the present invention, the term 'microprocessor' means also a central processing unit (CPU) or a digital signal processor (DSP), the meaning of these terms being the one commonly described in the literature. As usual, a microprocessor has an instruction set. In other words, the machine code of a program which is running or executed on said microprocessor, contains exclusively instructions belonging to said instruction set. Said machine code is obtained either by compiling the source code of said program or by manual writing. Each instruction of a said instruction set has an instruction format. As usual, the term 'instruction format' refers to a sequence of bit fields of a certain length. Said bit fields may be of different length. A minimum set of bit fields making up an instruction format normally contains a so called 'opcode' bit field and one or more 'operand' bit fields. Figure 1 illustrates the discussed concepts. The 'opcode' bit field encodes (allows to uniquely identify) a specific instruction, e.g. the addition of two numbers, among all the instructions of said instruction set. The 'operand' bit fields uniquely determine the operands of the instruction encoded in the 'opcode' bit field. In other words, an instruction is a data operation, where the operation is given by (encoded in) the 'opcode' bit field and where the data are given by (encoded in) the 'operand' bit fields. Usually, the operands are either given by memory references, e.g. data stored at some memory addresses, or by contents of registers in which case the registers are uniquely identified by (encoded in) said 'operand' bit fields. E.g. in case of a microprocessor with a register file containing 128 registers, an 'operand' bit field of at least 7 bits is required to uniquely identify (encode) a specific register inside the register file. Furthermore, one distinguishes normally between source operands and destination operands. Usually, source operands represent either memory references or registers containing the data required by an instruction, whereas destination operands represent either memory references or registers to which the result of an instruction, e.g. the addition of two numbers, has to be stored.

In the context of the present invention, the length and the order of the bit fields making up the format of an instruction is not relevant. In other words, it doesn't matter whether the 'opcode' bit field is preceding the 'operand' bit fields or vice versa nor does the order of the 'operand' bit fields among each other

matter. The encoding of the bit fields is not relevant as well. Finally, the number of operand bit fields is not relevant either.

Within the scope of the present invention, it is assumed that a microprocessor (CPUs or DSPs as well) operates with a basic clock and that, as is usual for today's microprocessors (CPUs and DSPs as well), instructions are pipelined. This means that said microprocessor has an instruction pipeline containing several stages and that instructions take several cycles of said clock to go through the different stages of the instruction pipeline before completing execution, the first pipeline stage being usually a 'prefetch' stage and the last pipeline stage being often a 'write back' or an 'execution' stage. Therefore, if a microprocessor operates with a basic clock, this means that data operations done inside said microprocessor as well as the depth of the instruction pipeline are given in cycle units of said clock. Typical depths of instruction pipelines of today's microprocessors range between 5 to 15 stages, in other words it takes from 5 up to 15 clock cycles for an instruction to go through the entire pipeline. Usually, each instruction has a different number of pipeline stages to go through. The number of pipeline stages that a given instruction has to go through is called the latency (in clock cycle units) of said instruction. Concerning the operation of a microprocessor and concerning the execution of a machine code on said microprocessor, a time axis can be defined by starting to count and label the clock cycles upwards, from a certain point in time onwards or when said microprocessor starts operation and begins to execute said machine code. If not mentioned otherwise, in the following the terms 'instruction scheduling' and 'instruction execution' refer to the definition and determination of the points on said time axis at which a given instruction within said machine code has to enter the different stages of the instruction pipeline. Note that this has not to be confounded with the instruction scheduling done by compiler techniques like software pipelining, list or trace scheduling etc... The point in time (on said time axis) at which a given instruction enters a pipeline stage is called the 'entrance point' of said instruction into said pipeline stage.

3. Prior Art

As mentioned before, a minimum set of bit fields making up an instruction format contains at least 'opcode' and 'operand' bit fields. Instruction formats of today's microprocessors, DSPs and CPUs may contain different flavors of said bit fields and usually contain additional bit fields as well.

First, instruction formats may be of fixed or of variable length and my contain a fixed number or a variable number of operands. In case of a variable instruction format length and a variable number of operands, additional bit fields may be spent for these purposes. However, format length and number of operands may also be part of the 'opcode' bit field.

Second, often an 'operand' bit field is given in form of an 'address specifier' bit field and an 'address' bit field. The 'address specifier' bit field determines the addressing mode for the considered operand, e.g.

indirect addressing, offset addressing etc..., whereas the 'address' bit field determines the address of the considered operand within a memory space.

However, none of today's instruction formats contains a bit field encoding explicit timing information, where said timing information explicitly determines instruction scheduling and execution as defined before. This lack of information is due to the fact that the architecture concepts of today's microprocessors (CPUs and DSPs as well) doesn't require this type of information because instruction scheduling is done either (1) in case of super-scalar and multi-issue microprocessors (CPUs and DSPs as well), by dynamic scheduling mechanisms based on data dependence analysis of instructions contained in a more or less large instruction window of the compiled or hand written machine code of a given program or (2) in case of VLIW processors by static scheduling techniques, in particular by software pipelining and trace scheduling, such that instructions are scheduled and executed in the same order in which they are arranged in the machine code, where said machine code is generated by applying said static scheduling techniques or (3) in case of EPIC processors, e.g. the IA-64 from Intel Corporation, by a mixture of the approaches (1) and (2). In this sense, timing information contained (encoded) in the instruction format appears to be just redundant information and only likely to increase the machine code size. However this does not hold in conjunction with a static instruction scheduling technique called software pipelining, as will be shown in section 5.

4. Brief description of the drawings

Figure 1 shows an example of a 'conventional' instruction format containing bit fields for 'opcode' and 'operands'.

Figure 2 shows an example of an instruction format containing a bit field containing explicit timing information.

Figure 3 shows a 'for'-loop and the directed acyclic graph ('dag') which equivalently represents the loop body of said 'for'-loop. Nodes of said 'dag' represent instructions of an instruction set of a microprocessor and where said 'dag' is 'software pipelined' with an initiation interval of 1 clock cycle.

5. Detailed description of the drawings

The main aspects of the present invention are described by referring to the figures mentioned in this section.

Figure 2 shows an instruction format containing a bit field with explicit timing information. Note that the position of the bit field within the instruction format is not relevant for the scope of the present invention. The main aspect of the present invention consists in introducing explicit timing information into instruction formats in general and to show the impacts on machine code size in conjunction with certain scheduling techniques. In the discussion that follows, it is assumed that the microprocessor for which such an instruction format is devised, operates with a basic clock. In other words, time indications referring to instruction scheduling and execution as well as the depth of the instruction pipelines are given in cycle units of said clock. Furthermore, a time axis is defined by starting to count and label the clock cycles upwards, from a certain point in time onwards or when said microprocessor starts operation or starts execution of some machine code. Furthermore, it is assumed that instructions are pipelined, in other words an instruction may take several clock cycles to go through all the stages of the instruction pipeline before completing execution. Furthermore, instructions may have different latencies as defined in the previous section.

Two problems related to instruction formats containing explicit timing information are now considered :

(1) given explicit timing information, how are the points on said time axis determined at which a given instruction has to enter a certain stage of the instruction pipeline

(2) how is the timing information encoded

To problem (1) : As mentioned before, it is natural to take as time unit the cycle of the basic clock of the microprocessor. As mentioned before, it is feasible that the timing information contained in the corresponding bit field of the instruction format of a given instruction contains timing information for each pipeline stage. In other words, the point in time at which a given instruction has to enter a certain pipeline stage, e.g. a 'decode' or an 'execution' stage, is contained in said timing information in form of a positive integer delay and said point in time (on said time axis) is obtained by adding said delay to the time reference of said instruction (this is called 'absolute timing' encoding) or to the point in time (on said time axis) at which said instruction entered a previous pipeline stage (this is called 'incremental timing' encoding). It is natural to take the point in time at which an instruction would enter the first pipeline stage in the absence of any timing (delay) information as time reference (called 'time zero'), for that instruction. However, the definition of the time reference is formal and any other pipeline stage may be considered as time reference as well.

An example shall illustrate the concepts. Consider an instruction pipeline of 3 stages consisting of 'fetch', 'decode' and 'execute' stages and assume that the bit field of the instruction format containing

explicit timing information for a given instruction contains the integers 2, 3 and 5. This would mean that said instruction would

(a) enter the 'fetch' stage with a delay of 2 clock cycle units with respect to 'time zero', where 'time zero' is the point in time or the clock cycle when the instruction would enter the 'fetch' stage in the absence of any delay information
(b) enter the 'decode' stage 3 clock cycles after having entered the 'fetch' stage
(c) enter the 'execute' stage 5 clock cycles after having entered the 'decode' stage.

As one can see, the timing information, given in form of positive integers, represents delays (in clock cycle units) according to which the entrance points of an instruction into the different pipeline stages have to be delayed with respect to points in time at which said instruction entered the previous pipeline stage. As explained before, the entrance point into the first pipeline stage is thereby delayed with respect to 'time zero', where 'time zero' is the point in time at which said instruction would enter the first pipeline stage in the absence of any timing (delay) information. Using a different terminology, one simply says that the entrance points must be delayed by the delays as given by the integer values contained in the timing information bit field of the instruction format. Therefore, it is assumed that the microprocessor contains some mechanism or hardware circuitry to delay the entrance points of an instruction into each pipeline stage individually. However, it is not relevant for the scope of the present invention how this mechanism is implemented, whether the delays are generated by stalls of the instruction pipeline or by some other method. In the previous example 'incremental timing' encoding was used, in other words the entrance point of an instruction into a certain pipeline stage is determined by adding the delay (as given by the integer value) to the entrance point into the previous pipeline stage. For the scope of the present invention, it must be noted that the method of delaying the entrance point of an instruction into a certain pipeline stage is equivalent to leaving the entrance point unchanged and delaying the point in time at which the instruction 'leaves' said pipeline stage, which is equivalent to increasing the latency of said pipeline stage, where the latency of a pipeline stage can be defined as the number of clock cycles that an instruction takes in order to go through said pipeline stage.

To problem (2) : As mentioned before, the timing information contained in the corresponding bit field of the instruction format may contain timing information for each pipeline stage of a given instruction. Although it is not relevant for the scope of the present invention, two basic encoding schemes are of practical interest and shall be briefly considered. Of course, there exists a myriad of encoding techniques allowing to further compress the timing information by minimizing the redundancy. This however always requires some decoding overhead prior to actual instruction scheduling and execution and usually implies some loss in overall processing speed performance as well as additional power consumption. The two mentioned encoding schemes which shall be considered here are : (a) 'absolute timing' (b) 'incremental timing'. 'Incremental timing' encoding has been used in the previous example. If 'absolute timing' encoding would be used instead, then said bit field would contain the integers 2, 5 (=2+3) and 10 (=2+3+5) respectively and all timing information would be with respect to the time reference (time 'zero') of said instruction, in other words the 'decode' stage would be entered 5 clock

the present discussion to specify the loop body in form of a graphical representation, namely in form of a directed acyclic graph (abbreviated by 'dag' in the following) whose nodes represent instructions of the instruction set of a given microprocessor. A directed edge emanating from a node v_1 and ending at a node v_2 means that node v_2 has to be scheduled and executed **after** node v_1. The presence of 3 nested 'if-then-else' statements in the loop body of the 'for'-loop translates into 3 'compare' instructions in the 'dag' and results in 4 possible branches such that one of the nodes labeled a, b, c or d in figure 3 are executed depending on the outcome of the 'compare' nodes labeled e, f and g. Assuming that there are no data dependencies between iterations of the 'for'-loop, the goal is now to maximize instruction level parallelism and to overlap the scheduling and execution of the different iterations of the 'for'-loop by applying software pipelining and determining the minimum initiation interval. Furthermore, assume that the resource constraints of the microprocessor are such that no more than three instructions can be scheduled and executed at the same time (in the same clock cycle). Neglecting any additional constraints due to operand (register) lifetimes, one can easily verify that the minimum initiation interval is 1 clock cycle long. Furthermore, the 'dag' shown in figure 3 is such that no instruction has to be delayed. However, since the 'dag' is software-pipelined with a period of one clock cycle, 3 independent 'compare' instructions have to be scheduled in one clock cycle leading to 2^3 possible combinations containing each 4 instructions taken from different iterations of the 'for'-loop, namely the combinations : (a,e,g,h), (a,f,g,h), (b,e,g,h), (b,f,g,h), (c,e,g,h), (c,f,g,h), (d,e,g,h), (d,f,g,h). This means that the final machine code corresponding to the 'software-pipelined' 'for'-loop contains at least $2^3 \times 4 = 32$ instructions (it effectively contains even more because additional 'branch' instructions must be inserted in the machine code), which is 4 times more than the number of instructions of a sequential machine code version of the 'for'-loop. Indeed, said sequential machine code version would contain only as many instructions as contained in the 'dag' under the assumption that predicated instructions would be used.

However, by using an instruction format with explicit timing information one can reduce the machine code size obtained by software-pipelining the considered 'dag' or 'for'-loop to the same number of instructions as required by said sequential machine code version. Indeed, it is enough

(1) to indicate the initiation interval of said 'dag' (e.g. of said 'for'-loop body)

(2) to indicate for each instruction (node) in the 'dag', the delay of that instruction such that all resource constraints are satisfied

This is enough information for an appropriately designed microprocessor to schedule and execute all instructions such that said 'for'-loop is effectively software-pipelined with the prescribed initiation interval. Although in the example of figure 3 all the delays are zero, it is easy to figure out this mechanism for the case where the delays are non-zero. As already mentioned previously, delays will usually use 'incremental timing' encoding. In other words, if the entrance point of a node (instruction) with no incoming edges is determined to be at some point on said time axis by taking into account the delay associated to that node, then any node v_2 having a associated delay d_2 and having an incoming edge emanating from some node v_1, where node v_1 entered the first pipeline stage at some clock cycle t (on said time axis) after taking into account the delay associated to that node, is then scheduled to enter the first pipeline stage at clock cycle $t+d_2+1$.

cycles after 'time zero' and the 'execution' stage 10 clock cycles after 'time zero'. As one can see, 'incremental timing' will normally require less bits to encode than 'absolute timing'.

The concept of 'incremental timing' and 'absolute timing' can be applied unchanged to as sequence of instructions which have to be scheduled and executed consecutively. Consider f. ex. a microprocessor containing an instruction pipeline with 3 stages. Consider an instruction i_1 containing timing information given in form of the integer delays 2, 3 and 5. Consider another instruction i_2, which has to be scheduled and executed consecutively to instruction i_1 and which contains timing information given in form of the integer delays 1, 2 and 3. Then if 'incremental timing' was used to encode the mentioned delays, it would mean that if instruction i_1 enters its 3 pipeline stages at clock cycles *t+2, t+5, t+10* respectively *(t* being the time reference for said instruction), then instruction i_2 enters its 3 pipeline stages at clock cycles *(t+2,t+5,t+10)+(1,2,3)= t+2+1, t+5+2, t+10+3* respectively.

One advantage of introducing timing information for each pipeline stage is to avoid hardware resource conflicts. E.g. consider the case of two instructions which are issued in parallel (in other words which enter the first pipeline stage at the same point in time), which have the same latencies and which must share the same ALU (Arithmetic Logic Unit) circuitry. Then, by delaying the entrance points into each pipeline stage appropriately, it is possible to avoid that the two instructions access the ALU at the same point in time or at the same clock cycle.

However, although the possibility of delaying entrance points individually allows for greater scheduling freedom, the special case in which the timing information contained in the bit field of the instruction format contains only one single delay is interesting as well. In this case, said delay specifies how much the entrance point of the given instruction into the first pipeline stage has to be delayed with respect to 'time zero', where as before 'time zero' is the point in time (or the clock cycle) when said instruction would enter the first pipeline stage in the absence of any delay. All consecutive pipeline stages are then entered without any additional delays.

E.g. assume that, in the absence of any timing information in the instruction format, an instruction would enter the pipeline stages at clock cycles *t, t+1, t+2* ... respectively, where *t* is the time reference for said instruction. Then if the instruction format of said instruction would contain timing information in the form of a single delay given by some integer *c*, this would imply that the pipeline stages would now be entered at clock cycles *t+c, t+c+1, t+c+2* ... respectively. In the case that the timing information contained in the instruction format of a given instruction contains is given in form of) only a single delay, one says that said delay is associated to said instruction.

Before closing this section, it is interesting to show the impact of incorporating explicit timing information in the instruction format on the machine code size of a given program. To this end we consider a simple example, shown in figure 3, of a 'for'-loop whose loop body contains three nested 'if-then-else' statements. Besides specifying the loop body in a high level language like C, it is more convenient for

Hence, by using an instruction format with explicit timing information, it is possible to keep the machine code size of the software-pipelined version of a 'for'-loop almost as compact as a sequential version thereof. Although a machine code size overhead is generated due to the additional bit-field in the instruction format containing the timing information, it will be small because in practice because the delays will lie in the range of a few clock cycles.

Finally as was mentioned before, it is assumed that the microprocessor for which such an instruction format with explicit timing information is designed, contains mechanisms and hardware circuitry

(a) to automatically start a new iteration every p clock cycles, where p represents the initiation interval (in clock cycle units), and to overlap the scheduling and execution of said iterations such that no hardware resource constraints are violated

(b) for each iteration, to delay the entrance points of the instructions into the instruction pipeline stages according to the timing information contained in the instruction format of said instructions such that all resource constraints, including register file constraints, of the given microprocessor are satisfied

Although the example in figure 3 and the previous discussion considers a 'for'-loop, the same methodology is applicable to loops in general including 'while'-loops. This is due to the fact that the loop body of any loop (whether 'for' or 'while'-loop) can be modeled by a 'dag'.

Before closing this section, it is important to note that the scope of the present invention covers as well the case in which the bit-field containing the timing information of an instruction is taken out of (or separated from) the instruction format and stored as a separate part (of the instruction program) which contains only timing information.

6. Summary of the invention

The present invention concerns a microprocessor having an instruction format containing explicit timing information according to claim 1.

Claims

What is claimed is :

1. A microprocessor having an instruction format containing explicit timing information,
   where said instruction format refers to all the instructions being part of the instruction set of said microprocessor,
   where said microprocessor contains an instruction pipeline containing one or more stages,
   where said machine code of said microprocessor contains exclusively instructions being part of said instruction set,
   where said microprocessor operates with a basic clock such that all time indications referring to instruction scheduling and execution as well as the depth of the instruction pipeline of said microprocessor are given in cycle units of said clock,
   where a time axis is defined by starting to count and label the clock cycles of said clock upwards from a certain point in time onwards or when microprocessor starts operation and begins to execute the machine code of a given program,
   where instructions, being part of said machine code which is executed on said microprocessor, are pipelined such that instructions take one or more clock cycles to go through one or more stages of the instruction pipeline before completing execution,
   where said timing information contained in the instruction format of an instruction contains one or more positive integer values representing delays according to which one or more entrance points (on said time axis) of said instruction into one or more pipeline stages have to be delayed either with respect to the point in time at which said instruction entered the previous pipeline stage or with respect to 'time zero' of said instruction, where the entrance point of said instruction into the first pipeline stage is delayed with respect to 'time zero', where 'time zero' is the point in time at which said instruction would enter the first pipeline stage in the absence of any delay,
   where said microprocessor contains some mechanism and hardware circuitry to delay the entrance points of the instructions into each pipeline stage according to the delays contained in the timing information of the instruction format

2. A microprocessor having an instruction format containing explicit timing information as claimed in claim 1., where said microprocessor contains mechanisms and hardware circuitry to software-pipeline loops, that is
   (a) to automatically start a new iteration of a given loop every p clock cycles, where p represents the initiation interval (in cycle units of said clock) and to overlap the scheduling and execution of said iterations of said loop
   (b) for each iteration of said loop, to delay the entrance points of the instructions into the stages of the instruction pipeline according to the timing information contained in the instruction format of said instructions such that all resource constraints of said microprocessor are satisfied

A microprocessor having an instruction format containing explicit timing information.

Abstract

The present invention describes an instruction format of a microprocessor (and of a CPU and DSP as well), said instruction format containing explicit timing information. Said timing information is specified in a dedicated bit-field and determines the delay in clock cycle units of said microprocessor by which the entrance point and subsequent decoding and execution of an instruction into the instruction pipeline of said microprocessor has to be delayed with respect to some predefined point in time. The advantages of the presence of such a timing information in the instruction format consists in substantially reducing the machine code size of software-pipelined 'for'-loops containing conditional statements such as 'if-then-else' statements.

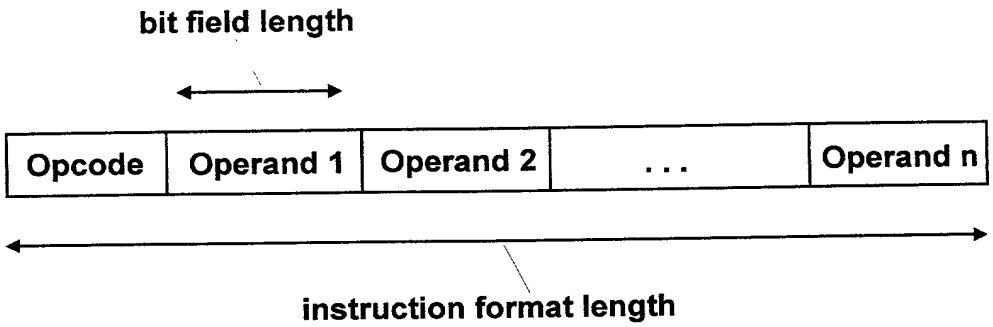


Figure 1

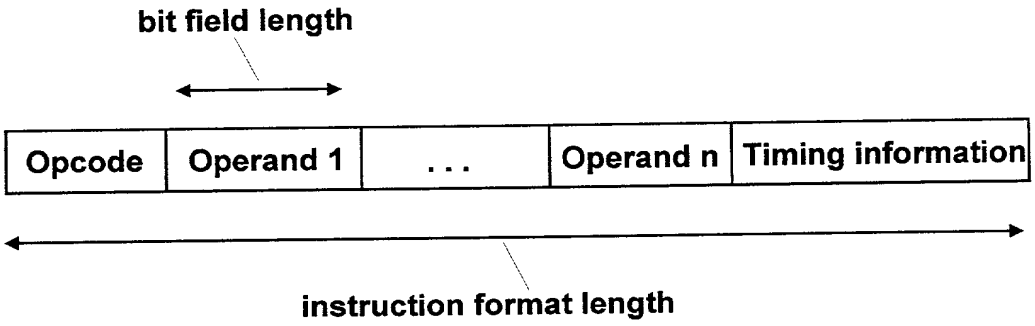
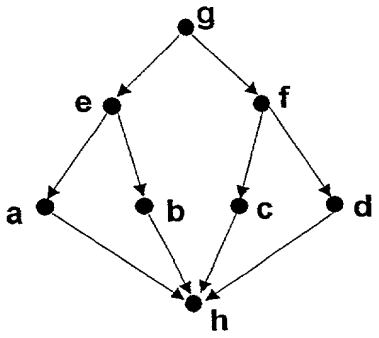


Figure 2

```
for (i==0;i<=N;i++)
{
if ( x[i]<=y[i] ) { if ( x[i+10]>=y[i] ) x[i]=y[i];
                    else x[i]=x[i+3]; }
  else { if ( x[i+3]>=y[i] ) x[i]=y[i+3];
         else x[i]=x[i+5]; }
}
```

loop body of the 'for'-loop



'dag' equivalently representing the loop body of the above 'for'-loop

Figure 3

PTO/SB/01 (10-01)
Approved for use through 10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION FOR UTILITY OR DESIGN PATENT APPLICATION (37 CFR 1.63)

☒ Declaration Submitted with Initial Filing OR ☐ Declaration Submitted after Initial Filing (surcharge (37 CFR 1.16 (e)) required)

Attorney Docket Number	
First Named Inventor	Jean-Paul Theis
COMPLETE IF KNOWN	
Application Number	PCT /EP00/07020
Filing Date	21 July 2000
Art Unit	
Examiner Name	

As the below named inventor, I hereby declare that:

My residence, mailing address, and citizenship are as stated below next to my name.

I believe I am the original and first inventor of the subject matter which is claimed and for which a patent is sought on the invention entitled:

A microprocessor having an instruction format containing explicit timing information

(Title of the Invention)

the specification of which

☒ is attached hereto

OR

☐ was filed on (MM/DD/YYYY) ____ as United States Application Number or PCT International

Application Number PCT/EP00/07020 and was amended on (MM/DD/YYYY) ____ (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment specifically referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56, including for continuation-in-part applications, material information which became available between the filing date of the prior application and the national or PCT international filing date of the continuation-in-part application.

I hereby claim foreign priority benefits under 35 U.S.C. 119(a)-(d) or (f), or 365(b) of any foreign application(s) for patent, inventor's or plant breeder's rights certificate(s), or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent, inventor's or plant breeder's rights certificate(s), or any PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application Number(s)	Country	Foreign Filing Date (MM/DD/YYYY)	Priority Not Claimed	Certified Copy Attached? YES	NO
			☐	☐	☐
			☐	☐	☐
			☐	☐	☐
			☐	☐	☐

☐ Additional foreign application numbers are listed on a supplemental priority data sheet PTO/SB/02B attached hereto:

PTO/SB/01 (10-01)
Approved for use through 10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION — Utility or Design Patent Application

Direct all correspondence to: ☒ Customer Number or Bar Code Label [ ] OR ☐ Correspondence address below

Name Ante Vista GmbH

Address Harburger Schlossstrasse 6-12

City Hamburg | State | ZIP 21079

Country Germany | Telephone +49 40 766 292661 | Fax

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

NAME OF SOLE OR FIRST INVENTOR : ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any]) Jean-Paul | Family Name or Surname Theis

Inventor's Signature | Date

Residence: City Hamburg DEX | State | Country Germany | Citizenship Luxembourg

Mailing Address Harburger Schlossstrasse 6-12

City Hamburg | State | ZIP 21079 | Country Germany

NAME OF SECOND INVENTOR: ☐ A petition has been filed for this unsigned inventor

Given Name (first and middle [if any]) | Family Name or Surname

Inventor's Signature | Date

Residence: City | State | Country | Citizenship

Mailing Address

City | State | ZIP | Country

☐ Additional inventors are being named on the ____supplemental Additional Inventor(s) sheet(s) PTO/SB/02A attached hereto.